没有关系数据库，可能就没有今天的IT和互联网产业。然而关系数据库在设计之初并没有预见到IT行业发展如此之快，总是假设系统运行在单机这一封闭的系统上。

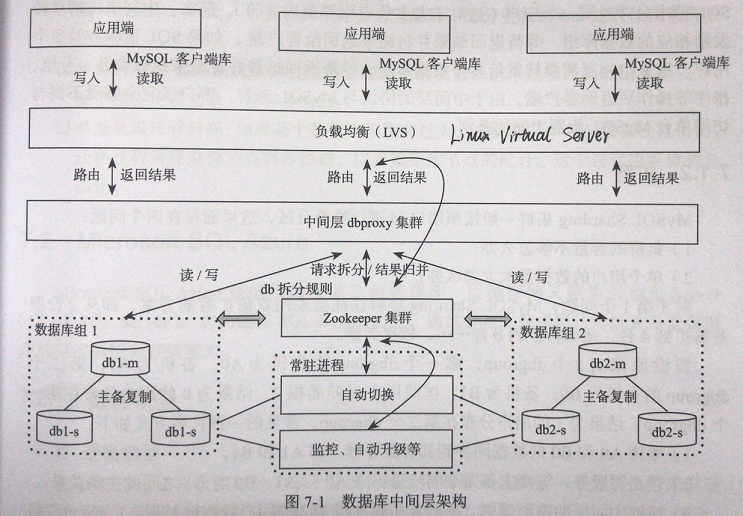
关系数据库的扩展有很多思路，比如：

1. 在应用层划分数据，将不同的数据分片划分到不同的关系数据库上，MySQL Sharding;
2. 关系数据库内部支持数据自动分片，Microsoft SQL Azure;
3. 全新的分布式数据库，Google spanner，Alibaba OceanBase。

# mysql sharding

## 架构

最简单也是最常见的做法就是在应用层按照规则将数据拆分，分布到多个数据库节点上，需要引入一个中间层对应用屏蔽后端数据库拆分的细节。



在mysql sharding架构中，有中间层dbproxy集群、数据库组、元数据服务器、常驻进程这些部分。

1. dbproxy中间层

中间层解析客户端的sql请求并转发到后端的数据库。它要负责解析sql语句、执行sql路由、结果归并、排序以及分组等，中间层由多个无状态的dbproxy进程组成。可以在客户端和中间层之间引入LVS对客户端请求进行负载均衡，但是LVS本身有一层通信开销，所以常见的做法是直接在客户端配置中间层服务器的列表，让客户端自己处理请求负载均衡以及中间层故障等问题。

1. dbgroup数据库组

每个dbgroup由N台数据库机器组成，其中一台为master，剩下的为slave。master负责所有的写事务及强一致读事务，并将操作以binlog的形式复制到slave。

1. 元数据服务器

元数据服务器主要负责dbgroup拆分规则并用于dbgroup选主。dbproxy通过元数据服务器获取拆分规则从而确定SQL语句的执行计划。dbgroup的选主也需要通过元数据服务器进行。元数据本身也需要多个副本实现HA（High Available），常见的方式是采用Zookeeper。

1. 常驻进程

部署在**每台数据库服务器**上的常驻进程，用于实现监控、单点切换、安装和卸载程序等。dbgroup中的数据库需要进行准备切换，这些控制逻辑需要与数据库读写事务处理逻辑隔离开。

如果数据库安装用户ID哈希分区，那么同一个用户的数据分布在一个dbgroup上。如果一个sql请求只涉及到一个用户，那么中间层将请求转发给对应的一个dbgroup，等待返回的结果转发给客户端就可以了；如果sql涉及到多个用户，那么中间层需要转发给多个dbgroup，等待返回的结果执行合并、分组、排序等操作，然后再返回给客户端。由于中间层的协议与mysql兼容，客户端完全感受不到与访问单台mysql机器之间的差别。

## 扩容

mysql sharding集群通常采用双倍扩容的方案。假设有2个dbgroup，第一个dbgroup有主机a0和a1，第二个dbgroup有主机b0和b1。所以当前的状况是按照用户id哈希对2取模，按照哈希结果选择对应的dbgroup。

现在对这个集群进行扩容：

1. 等待a0和b0的数据分别同步a1和b1；
2. **停止写服务**，解除a0与a1，b0与b1之间的主备关系；
3. 修改中间层的映射规则，将哈希对4取模结果是2的用户数据映射到a1，结果是3的映射到b1；
4. 开启写服务；
5. 分别给a0、a1、b0和b1增加一台备份机。

这样下来，原来2个dbgroup变成了4个dbgroup。可以看到扩容的过程中集群需要停一会服务。

对于单个用户数据量过大的情况，可以在应用层定期统计出大用户，并且这些用户的数据按照数据量拆分到多个dbgroup。定期维护这些信息对应用层来说是一个很大的代价。

## 问题

引入数据库中间层将后端分库分表对应用透明化，在大型互联网公司内部很常见。但是也面临一些问题：

1. 数据库复制：mysql主备之间只支持异步的复制，而且主库压力较大时可能产生很大的延迟，因此主备切换时可能会丢失最后一部分更新事务。
2. 扩容问题：增加新机器涉及到数据的重新划分，过程比较复杂，容易出错。
3. 动态数据迁移问题：某个dbgroup压力过大时，需要将其中部分数据迁移出去，迁移过程需要总控节点整体协调，很难做到自动化。

# Azure

Azure是云关系型数据库，是构建在SQL server之上，通过分布式技术提升传统关系型数据库的可扩展性和容错能力。

# Google Spanner

# OceanBase

## 背景

淘宝收藏夹应用，包含info表和item表，info表是收藏信息的条目，item是收藏的宝贝和店铺的详细信息。

如果用户按照商品价格排序，那么要先从item表中读取收藏物品的价格等最新的信息，耗时会比较大；如果要把物品的详细信息冗余到info表，那么就不需要从item取数据了，但是许多商品有非常多的人收藏，这些商品价格的改动会导致info表大量的修改。

## 设计思路

数百TB的数据量、数十万TPS的访问量，采用任何单台的关系型数据库都无法应对，一张做法是根据业务特点对数据库水平拆分，比如根据用户id哈希取模后分布到不同的数据库，再加一个中间层路由到不同的分区，但是这种方式还是有很多的弊端：

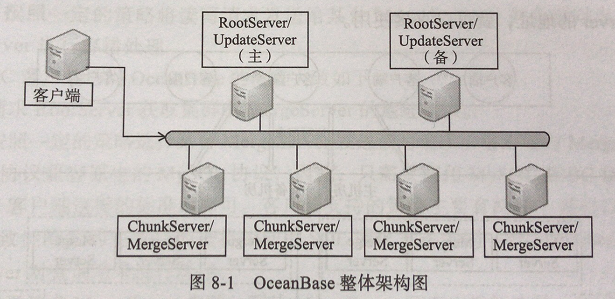
1. 数据量和负载增加后想要添加机器操作很难，需要人工介入。（可以采用一致性哈希）
2. 有些sql语句需要访问几乎所有的分区，比如是按照用户id分区，查询收藏了一个商品的用户需要访问所有的分区；

另一种做法是参考分布式表格的做法，将大表划分成很多个子表，子表之间按照主键有序，某台服务器发送故障，它上面服务的数据能够在很短的时间内自动迁移到集群中其他的服务器。这种方式解决了可扩展性的问题，少量的突发事件对使用者是透明的。这种分布式的做法虽然解决了扩展性的问题，但是却无法支持事务。比如BigTable只支持单行事务，OceanBase希望能够支持跨行跨表事务。

最直接的做法是在BigTable开源实现（HBase或者Hypertable）的基础上引入两阶段提交协议支持分布式事务，但是这种方式对事务的响应时间也是很高的，Percolator中事务的平均响应时间在2-5秒。

虽然在线业务的数据量十分庞大，上百亿条，但是最近一段时间修改量并不多，几千万或者上亿条，OceanBase采用单台更新服务器来记录最近一段时间的修改增量，而之前的数据保持不变，之前的数据成为基线数据。基线数据以类似分布式文件系统的方式存储在多台基线数据服务器中，每次查询要把基线数据和增量数据融合后返回给客户端。写事务都集中在单台更新服务器上，避免了复杂的分布式事务，高效的实现了跨行跨表事务。更新服务器上的修改增量也会定期发给多台基线数据服务器中，避免成为瓶颈。

## 系统架构



### client

类似mysql的JDBC操作；

### RootServer

主要功能包括：集群管理、数据分布以及副本管理。管理集群中所有的MergeServer、ChunkServer以及UpdateServer。

一个集群内部同一时刻只允许一个UpdateServer提供写服务，称为主UpdateServer，牺牲了一定的可用性获取了强一致性，RootServer通过租约机制选择唯一的主UpdateServer。

RootServer采用一主一备的结构，主备之间数据强同步，通过Linux HA软件实现高可用。

### UpdateServer

存储系统中增量更新数据。UpdateServer也是一主一备，主备之间可以配置不同的同步模式。部署时，UpdateServer和RootServer进程共用物理服务器。

UpdateServer是集群中唯一能够接受写入的模块，每个集群只有一个主UpdateServer。UpdateServer中的更新操作首先写入到内存表，当内存表数据量过大时生成快照文件转存到SSD中。

为保证可靠性，主UpdateServer更新内存表前先写操作日志，并同步到备UpdateServer。由于集群中只有一台主UpdateServer提供写服务，因此OceanBase很容易实现跨行跨表事务，而不需要采用传统的两阶段提交协议。但是这样集群中所有的读写操作都必须经过UpdateServer。OceanBase通过定期合并和数据分发这两种机制将UpdateServer一段时间之后的增量更新源源不断的分散到ChunkServer，而UpdateServer只需要服务最新一小段时间新增的数据，这些数据往往可以全部存放在内存中。

### ChunkServer

存储系统中的基线数据。

### MergeServer

**接收并解析用户sql请求**，对sql请求做一系列优化后传给相应的ChunkServer或者UpdateServer。如果请求的数据分布在多台ChunkServer，MergeServer还需要对多台ChunkServer返回的结果合并。